

# Optimal Continuous Time Markov Decisions

Yuliya Butkova, Hassan Hatefi, Holger Hermanns, Jan Krčál

Saarland University – Computer Science, Saarbrücken, Germany

**Abstract.** In the context of Markov decision processes running in continuous time, one of the most intriguing challenges is the efficient approximation of finite horizon reachability objectives. A multitude of sophisticated model checking algorithms have been proposed for this. However, no proper benchmarking has been performed thus far.

This paper presents a novel and yet simple solution: an algorithm, originally developed for a restricted subclass of models and a subclass of schedulers, can be twisted so as to become competitive with the more sophisticated algorithms in full generality. As the second main contribution, we perform a comparative evaluation of the core algorithmic concepts on an extensive set of benchmarks varying over all key parameters: model size, amount of non-determinism, time horizon, and precision.

## 1 Introduction

Over the last two decades, a formal approach to quantitative performance and dependability evaluation of concurrent systems has gained maturity. At its root are continuous-time Markov chains (CTMC) for which efficient and quantifiably precise solution methods exist [2]. A CTMC can be viewed as a labelled transition system (LTS) whose transitions are delayed according to exponential distributions. CTMCs are stochastic processes and thus do not support non-determinism. Non-determinism, often present in classical concurrency and automata theory models, is useful for modelling uncertainty or for performing optimisation over multiple choices. The genuine extension of CTMCs with non-determinism are continuous time Markov decision processes (CTMDPs). The non-determinism is controlled by an object called *scheduler* (also policy or strategy).

Prominent applications of CTMDPs include power management and scheduling [27], networked, distributed systems [10,16], epidemic and population processes [20], economy [5] and others. Moreover, CTMDPs are the core semantic model [8] underlying formalisms such as generalised stochastic Petri nets [21], Markovian stochastic activity networks [22] and interactive Markov chains [17].

When model checking a CTMDP [6], one asks whether the behaviour of the model for *some* schedulers (if we control the non-determinism) or for *all* schedulers (if it is out of control) satisfies given performance or dependability criteria. A large variety of them can be expressed using logics such as CSL [1]. At the centre of model-checking problems for such criteria is time bounded reachability: *What is the maximal/minimal probability to reach a given set of states within a*

*given time bound.* Having an efficient approach for this optimisation (maximisation or minimisation) is crucial for successful large-scale applications.

In order to not discriminate against real situations, one usually assumes that the scheduler can base its decisions on *any* available information about the past. Restricting the information however tends to imply cheaper approximative algorithms [25,3]. For CTMDPs, we can distinguish (general) *timed* optimal scheduling and (restricted) *untimed* optimal scheduling [3,4]. In the latter case, the scheduler has no possibility, intuitively speaking, to look at a clock measuring time. Another distinction within timed optimality discussed in the literature is *early* optimal scheduling (where every decision is frozen in between state changes [26,15]) and *late* optimal scheduling (where every decision can change as time passes while residing in a state [7,9]).

A handful of sophisticated algorithms have been suggested for timed optimality (partly for early optimality, partly for late optimality) signifying both the importance and the difficulty of this problem [26,7,9]. This paper presents a substantially different algorithm addressing this very problem. The approach is readily applicable to both early and late optimality. It harvests a very efficient algorithm for untimed optimality [3] originally restricted to a subclass of models. By a simple twist, we make it applicable for the general timed optimality for arbitrary models. As a second contribution, we present an exhaustive empirical comparison of this novel algorithm with all other published algorithms for the (early or late) timed optimality problem. We do so on an extensive collection of scalable industrial and academic CTMDP benchmarks (that we also make available). Notably, all earlier evaluations did compare at most two algorithms on at most one or two principal cases. We instead cross-compare 5 algorithms on 7 application cases, yielding a total of about 2350 distinct configurations. The results demonstrate that our simple algorithm is highly efficient across the entire spectrum of models, except for some of the experiments where extreme precision is required. On the other hand, no algorithm is consistently dominating any other algorithm across the experiments performed.

*Related work.* Timed optimal scheduling has been considered for many decades both theoretically [23,29] and practically by introducing approximative algorithms. Formal error bounds needed for verification have been studied only recently [26,15,9,7]. Fragmentary empirical evaluations of some of the published algorithms have been performed [6,9,15]. In a nutshell, the published knowledge boils down to [26]  $\prec_{[15]}$  [15] and [26]  $\prec_{[6]}$  [7]  $\prec_{[9]}$  [9], where  $a \prec_{[b]}$   $b$  denotes “ $b$  is shown empirically faster than  $a$  in  $[b]$ ”. A substantial cross-comparison of the newest three algorithms [7,9,15] is however lacking.

*Contribution of the paper.* The paper (i) develops a novel and simple approximation method for time bounded CTMDP reachability, (ii) presents the first ever set of benchmarks for CTMDP model checking, and (iii) performs an empirical evaluation across benchmarks and algorithms. The evaluation suggests that the optimal timing of decisions for time bounded reachability can be solved effectively by a rather straightforward algorithm, unless extreme precision is needed.

## 2 Preliminaries

**Definition 1.** A continuous-time Markov decision process (CTMDP) is a tuple  $\mathcal{C} = (S, \text{Act}, \mathbf{R})$  where  $S$  is a finite set of states,  $\text{Act}$  is a finite set of actions, and  $\mathbf{R} : S \times \text{Act} \times S \rightarrow \mathbb{R}_{\geq 0}$  is a rate function.

We call an action  $a$  *enabled* in  $s$ , also denoted by  $a \in \text{Act}(s)$ , if  $\mathbf{R}(s, a, s') > 0$  for some  $s' \in S$ . We require that all sets  $\text{Act}(s)$  are non-empty. A continuous-time Markov chain (CTMC) is a CTMDP where all  $\text{Act}(s)$  are singleton sets.

For a given state  $s$  and action  $a \in \text{Act}(s)$ , we denote by  $E(s, a) = \sum_{s'} \mathbf{R}(s, a, s')$  the *exit* rate of  $a$  in  $s$ . Finally, we let  $\mathbf{P}(s, a, s') := \mathbf{R}(s, a, s')/E(s, a)$ .

The operational behaviour of a CTMDP is like in a CTMC. Namely, when performing a given action  $a_0$  in a state  $s_0$ , the CTMDP waits for a transition, i.e. waits for a delay  $t_0$  chosen randomly according to an exponential distribution with rate  $E(s_0, a_0)$ . The transition leads to a state  $s_1$  again chosen randomly according to the probability distribution  $\mathbf{P}(s_0, a_0, \cdot)$ . When performing an action  $a_1$  there, it similarly waits for time  $t_1$  and makes a transition into a state  $s_2$  and so on, forming an infinite *run*  $s_0 t_0 s_1 t_1 \dots$ .

The difference to a CTMC lies in the need to choose actions to perform, done by a *scheduler*. There are two classes of schedulers, *early* and *late*. Whenever entering a state, an early scheduler needs to choose and commit to a next action, whereas late schedulers may change such choices at any time later while residing in the state. In this paper we restrict w.l.o.g. [24] to deterministic schedulers but we allow the decision to depend on the whole *history*  $s_0 t_0 \dots t_{n-1} s_n$  so far.

**Definition 2.** A (timed late) randomised scheduler<sup>1</sup> is a measurable<sup>1</sup> function  $\sigma$  that to any history  $h = s_0 t_0 \dots t_{n-1} s_n$  and time  $t \geq 0$  spent in  $s_n$  so far assigns a distribution over enabled actions  $\text{Act}(s_n)$ . We call  $\sigma$  *early* if  $\sigma(h, t) = \sigma(h, t')$  for all  $h, t, t'$ ; and *deterministic* if  $\sigma(h, t)$  assign 1 to some action  $a$  for all  $h, t$ .

We denote the set of all (timed) late or early schedulers by  $\text{Tim}_\ell$  and  $\text{Tim}_e$ , respectively. We use these subscripts  $\nabla \in \{\ell, e\}$  throughout the paper to distinguish between the late and the early setting. Furthermore, a scheduler  $\sigma$  is called *untimed* if  $\sigma(h, t) = \sigma(h', t')$  whenever  $h$  and  $h'$  contain the same sequence of states. By  $\text{Unt}$  we denote the set of all untimed schedulers. Note that  $\text{Unt} \subseteq \text{Tim}_e \subseteq \text{Tim}_\ell$ .

Fixing a scheduler  $\sigma$  and an initial state  $s$  in a CTMDP  $\mathcal{C}$ , we obtain the unique probability measure  $\text{Pr}_\sigma^{\mathcal{C}, s}$  over the space of all runs by standard definitions [24], denoted also by  $\text{Pr}_\sigma^s$  when  $\mathcal{C}$  is clear from context.

**Problem 1 (Maximum Time-Bounded Reachability)** Let  $\mathcal{C} = (S, \text{Act}, \mathbf{R})$ ,  $G \subseteq S$  be a set of goal states,  $T \in \mathbb{R}_{\geq 0}$  a time bound, and  $\nabla \in \{\ell, e\}$ . Approximate the values  $\text{val}_\mathcal{C}^\nabla \in [0, 1]^S$ , where each  $\text{val}_\mathcal{C}^\nabla(s)$  maximises the probability

$$\text{val}_\mathcal{C}^\nabla(s) := \sup_{\sigma \in \text{Tim}_\nabla} \text{Pr}_\sigma^s [\Diamond^{\leq T} G]$$

of runs  $\Diamond^{\leq T} G = \{s_0 t_0 \dots \mid \exists i : s_i \in G \wedge \sum_{j=0}^{i-1} t_j \leq T\}$  reaching  $G$  before  $T$ .

<sup>1</sup> Measurable with respect to the standard  $\sigma$ -algebra on the set of finite histories [24].

Whenever  $\mathcal{C}$  is clear from context, we write  $\text{val}^\nabla$ . We call  $\sigma \in \text{Tim}_\nabla$   $\epsilon$ -optimal if  $\Pr_\sigma^s [\Diamond^{\leq T} G] \geq \text{val}^\nabla(s) - \epsilon$  for all  $s \in S$ , and *optimal* if it is 0-optimal.

By minor changes, all results of the paper also address the dual problem of *minimum* time bounded reachability that we omit to simplify the presentation.

*Remark 1.* There exists a value preserving encoding of early scheduling into late scheduling in CTMDPs [28]. It has exponential space complexity (due to the number of induced transitions). This exponentiality does arise in practice, e.g. for the stochastic job scheduling problem considered later. Therefore we treat the two algorithmic settings separately. Early scheduling is natural for models derived from generalised stochastic Petri nets or interactive Markov chains.

### 3 Unif<sup>+</sup>: Optimal Time-Bounded Reachability Revisited

In this section, we develop a novel and simple algorithm for Problem 1. We fix  $\mathcal{C} = (S, \text{Act}, \mathbf{R})$ ,  $G \subseteq S$ ,  $T \in \mathbb{R}_{\geq 0}$ ,  $\nabla \in \{\ell, e\}$  and an approximation error  $\epsilon > 0$ . Furthermore, let  $E_{\max} := \max_{s,a} E(s, a)$  denote the maximal exit rate in  $\mathcal{C}$ .

In contrast to existing methods, our approach does not involve discretisation. The algorithm instead builds upon *uniformisation* [18] and *untimed* analysis [3,4,29]. It is outlined in Algorithm 1. Technically, it is based on an iterative computation of tighter and tighter lower and upper bounds on the values until the required precision is met. In the first iteration, a *uniformisation rate*  $\lambda$  is set to  $E_{\max}$ , in every further iteration its value is doubled. In every iteration, we compute a lower bound  $\underline{\text{val}}$  and an upper bound  $\overline{\text{val}}$  by two types of untimed analyses on the CTMDP  $\mathcal{C}_\lambda^\nabla$  obtained by uniformising  $\mathcal{C}$  to the rate  $\lambda$ . In the remainder of this section, we explain the individual steps of Algorithm 1, and prove correctness and termination.

Informally, the lower bound is based on maximum time bounded reachability with respect to the untimed scheduler subclass [3]. The upper bound, similarly to

---

#### Algorithm 1: UNIF<sup>+</sup>

---

**input** : CTMDP  $\mathcal{C} = (S, \text{Act}, \mathbf{R})$ , goal states  $G \subseteq S$ , horizon  $T \in \mathbb{R}_{>0}$ , scheduler class  $\nabla \in \{\ell, e\}$ , and approximation error  $\epsilon > 0$   
**params**: truncation error ratio  $\kappa \in (0, 1)$   
**output** : vector  $\mathbf{v}$  such that  $\|\mathbf{v} - \text{val}^\nabla\|_\infty \leq \epsilon$  and  $\lambda$

- 1  $\lambda \leftarrow$  maximal exit rate  $E_{\max}$  in  $\mathcal{C}$
- 2 **repeat**
- 3     $\mathcal{C}_\lambda^\nabla \leftarrow$   $\nabla$ -uniformisation of  $\mathcal{C}$  to the rate  $\lambda$
- 4     $\underline{\mathbf{v}} \leftarrow$  approximation of the lower bound  $\underline{\text{val}}$  for  $\mathcal{C}_\lambda^\nabla$  up to error  $\epsilon \cdot \kappa$
- 5     $\overline{\mathbf{v}} \leftarrow$  approximation of the upper bound  $\overline{\text{val}}$  for  $\mathcal{C}_\lambda^\nabla$  up to error  $\epsilon \cdot \kappa$
- 6     $\lambda \leftarrow 2 \cdot \lambda$
- 7 **until**  $\|\overline{\mathbf{v}} - \underline{\mathbf{v}}\|_\infty \leq \epsilon \cdot (1 - \kappa)$
- 8 **return**  $\underline{\mathbf{v}}, \lambda$

---

the one in [7], is based on *prophetic* untimed schedulers that yield higher value than timed schedulers by knowing in advance how many steps will be taken within time  $T$ . The intuition is that an untimed scheduler can approximately observe the elapse of time by knowing the count of steps taken and the expected delay per every step. In uniformised models, these delay expectations are identical across all states (forming a Poisson process) and therefore allow easy access to the expected total elapsed time. By uniformising the model with higher and higher uniformisation rates, this implicit knowledge of untimed schedulers increases. On the other hand, the knowledge of prophetic untimed schedulers decreases; both approaching the power of timed schedulers.

### 3.1 Uniformisation to $\mathcal{C}_\lambda^\nabla$

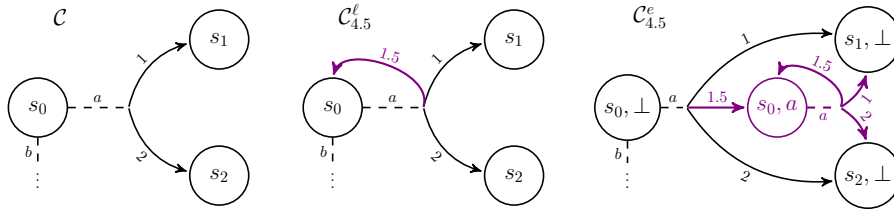
CTMDP  $\mathcal{C}$  may have transitions with very different rates across different states and actions. Here, we discuss how to perform *uniformisation* for such a model. This is a conceptually well-known idea [18]. Applying it to  $\mathcal{C}$  intuitively makes transitions occur with a higher rate  $\lambda \geq E_{max}$ , uniformly across all states and actions.

To ensure that uniformisation does not change the schedulable behaviour, we need distinct uniformisation procedures for the early and the late setting. Late uniformisation is straightforward, it adds self-loops to states and actions where needed.

**Definition 3 (Late uniformisation).** For  $\lambda \geq E_{max}$  we define the late uniformisation of  $\mathcal{C}$  to rate  $\lambda$  as a CTMDP  $\mathcal{C}_\lambda^\ell = (S, Act, \mathbf{R}_\lambda^\ell)$  where

$$\mathbf{R}_\lambda^\ell(s, a, s') := \begin{cases} \mathbf{R}(s, a, s') & \text{if } s \neq s', \\ \lambda - \sum_{s'' \neq s} \mathbf{R}(s, a, s'') & \text{if } s = s'. \end{cases}$$

*Example 1.* For the fragmentary CTMDP  $\mathcal{C}$  depicted below on the left, its late uniformisation to rate 4.5 is depicted in the middle.



Using the same transformation for the early setting would give the scheduler the spurious possibility to “reconsider” the choice of the action in a state whenever a newly added self-loop is taken. To exclude that possibility, early uniformisation introduces a copy state  $(s, a)$  for each state  $s$  and action  $a$  so as to “freeze” the commitment of choosing action  $a$  until the next state change occurs. The construction is shown on the right. States of the form  $(s, \perp)$  correspond to the original states, i.e. those where no action has been committed to yet.

**Definition 4 (Early uniformisation).** For  $\lambda \geq E_{max}$ , the early uniformisation of  $\mathcal{C}$  to rate  $\lambda$  is a CTMDP  $\mathcal{C}_\lambda^e = (S \times (\{\perp\} \cup \text{Act}), \text{Act}, \mathbf{R}_\lambda^e)$  where for every state  $(s, \cdot)$ , action  $a \in \text{Act}$ , and every successor state  $(s', \circ)$  we have

$$\mathbf{R}_\lambda^e((s, \cdot), a, (s', \circ)) := \begin{cases} \mathbf{R}(s, a, s') & \text{if } \circ = \perp, \\ \lambda - E(s, a) & \text{if } \circ = a, s = s', \\ 0 & \text{elsewhere.} \end{cases}$$

Uniformisation preserves the value of time-bounded reachability for both early [24] and late schedulers [23].

**Lemma 1.**  $\forall \lambda \geq E_{max}. \text{val}_{\mathcal{C}}^\nabla = \text{val}_{\mathcal{C}_\lambda^\nabla}^\nabla$ , i.e. uniformisation preserves the value.

As a result, we can proceed by bounding the values of  $\mathcal{C}_\lambda^\nabla$  for large enough  $\lambda$  instead of bounding the values of the original CTMDP  $\mathcal{C}$ .

### 3.2 Lower and upper bounds on the value of $\mathcal{C}_\lambda^\nabla$

We now fix a  $\lambda$  and consider a uniform CTMDP  $\mathcal{C}_\lambda^\nabla$ . We denote by  $\diamond_{\leq i}^{\leq T} G$  the subset of runs  $\diamond^{\leq T} G$  reaching the target where exactly  $i$  steps are taken up to time  $T$ . With this, we define the bounds by ranging over *Unt* schedulers in  $\mathcal{C}_\lambda^\nabla$ :

$$\underline{\text{val}}(s) := \sup_{\sigma \in \text{Unt}} \sum_{i=0}^{\infty} \Pr_\sigma^s [\diamond_{\leq i}^{\leq T} G], \quad \overline{\text{val}}(s) := \sum_{i=0}^{\infty} \sup_{\sigma \in \text{Unt}} \Pr_\sigma^s [\diamond_{\leq i}^{\leq T} G].$$

Since all  $\diamond_{\leq i}^{\leq T} G$  are disjoint and  $\diamond^{\leq T} G = \bigcup_{i \in \mathbb{N}_0} \diamond_{\leq i}^{\leq T} G$ , the value  $\underline{\text{val}}$  is the optimal reachability probability of standard untimed schedulers on the uniformised model. It will serve as a lower bound on the values  $\text{val}^\nabla$ . The value  $\overline{\text{val}}$ , on the other hand, which has the supremum and summation swapped, does not correspond to the value of any realistic scheduler. Intuitively, it is the value of a *prophetic* untimed scheduler, which for each particular run knows how many steps will be taken (as for every  $i$ , a different standard scheduler  $\sigma$  may be used). This knowledge makes the scheduler more powerful than any other timed one:

**Lemma 2.** It holds that  $\text{val}^e \leq \text{val}^\ell$ , and for any CTMDP  $\mathcal{C}_\lambda^\nabla$ ,  $\underline{\text{val}} \leq \text{val}^\nabla \leq \overline{\text{val}}$ .

**Approximating the bounds.** Since  $\underline{\text{val}}$  and  $\overline{\text{val}}$  are defined via infinite summations, we need to approximate these bounds. We do so by iterative algorithms truncating the sums. This is what is computed in line 4 and 5 of Algorithm 1. Each truncation induces an error of up to  $\varepsilon \cdot \kappa$ .

Let  $\psi_\lambda(k)$  denote the Poisson distribution with parameter  $\lambda T$  at point  $k$ , i.e. the probability that exactly  $k$  transitions are taken in the CTMDP  $\mathcal{C}_\lambda^\nabla$  before time  $T$ . Furthermore, let  $N = \lceil \lambda T e^2 - \ln(\varepsilon \cdot \kappa) \rceil$ , where  $e$  is the Euler's number. We recursively define for every  $0 \leq k \leq N$  and every state  $s$ , functions

$$\underline{v}_k(s) = \begin{cases} 0 & \text{if } k = N, \\ \sum_{i=k}^{N-1} \psi_\lambda(i) & \text{if } k < N \text{ and } s \in G, \\ \max_a \sum_{s'} \mathbf{P}_\lambda^\nabla(s, a, s') \cdot \underline{v}_{k+1}(s') & \text{if } k < N \text{ and } s \notin G, \end{cases}$$

$$\bar{w}_k(s) = \begin{cases} 0 & \text{if } k = N, \\ 1 & \text{if } k < N \text{ and } s \in G, \\ \max_a \sum_{s'} \mathbf{P}_\lambda^\nabla(s, a, s') \cdot \bar{w}_{k+1}(s') & \text{if } k < N \text{ and } s \notin G, \end{cases}$$

$$\bar{v}_k(s) = \sum_{i=k}^{N-1} \psi_\lambda(i) \cdot \bar{w}_{(N-1)-(i-k)}(s),$$

where  $\mathbf{P}_\lambda^\nabla$  denotes the transition probability matrix of  $\mathcal{C}_\lambda^\nabla$ .

**Lemma 3.** *In any CTMDP  $\mathcal{C}_\lambda^\nabla$ ,  $\|\underline{v}_0 - \underline{\text{val}}\|_\infty \leq \varepsilon \cdot \kappa$  and  $\|\bar{v}_0 - \bar{\text{val}}\|_\infty \leq \varepsilon \cdot \kappa$ .*

We compute  $\underline{v}_0$  as in the untimed analysis of uniform models [3], which in turn agrees with the standard “uniformisation” algorithm for CTMCs when the maximisation is dropped. The computation of  $\bar{w}_k$  is analogous to step-bounded reachability for *discrete-time* Markov decision processes, where the reachability probabilities for different step-bounds are weighted by the Poisson distribution in the end in  $\bar{v}_0$ . Both vectors can be computed in time  $O(N \cdot |S|^2 \cdot |\text{Act}|)$ .

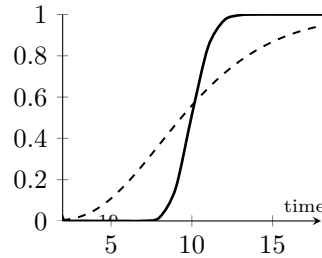
*Numerical Aspects.* In practice also  $\underline{v}_0$  and  $\bar{v}_0$  can only be approximated due to presence of  $\psi_\lambda(k)$ . For details how the overall error bound is met in an analogous setting, see [4]. For high values of  $\lambda$  and thus also  $N$ , the Poisson values  $\psi_\lambda(k)$  are low for most  $0 \leq k < N$  and also the values in  $\mathbf{P}_\lambda^\nabla$  get close to 1 when on the diagonal and to 0 when off-diagonal. Where high precision is required and thus high  $\lambda$  may be needed, attention has to be paid to numerical stability.

### 3.3 Convergence of the bounds for increasing $\lambda$

An essential part for the correctness of Algorithm 1 is its convergence:

**Lemma 4.** *We have  $\lim_{\lambda \rightarrow \infty} g_\lambda \rightarrow 0$  where  $g_\lambda$  denotes the gap  $\|\underline{\text{val}} - \bar{\text{val}}\|_\infty$  in  $\mathcal{C}_\lambda^\nabla$ .*

*Proof Idea.* We here provide an intuition of the core of the proof, namely why uniformisation with higher  $\lambda$  increases the power of untimed schedulers and decreases the power of prophetic ones: The count of transitions taken so far gives untimed schedulers approximate knowledge of how much time has elapsed. In situations with the *same expectation* of elapsed time, a higher uniformisation rate induces a *lower variance* of elapsed time. On the right, we illustrate comparable situations for different uniformisation rates, after 5 transitions with rate 0.5 and after 100 transitions with rate 10. Both depicted cumulative distribution functions of elapsed time have expectation 10 but the latter is way steeper, providing a more precise knowledge of time.



At the same time prophetic schedulers on the high-rate uniformised model are less powerful than on the original one. When taking decisions, the future evolution is influenced by two types of randomness: (a) continuous timing, i.e. how

many further transitions will be taken before the time horizon and (b) discrete branching, i.e. which transitions will be taken. Even though the value stays the same for arbitrary  $\lambda$ , the “source of” randomness for high  $\lambda$  shifts from (a) to (b). Namely, the distribution of the number of future transitions also becomes steeper for higher  $\lambda$ , thus being “less random” by having smaller coefficient of variation. At the same time, the discrete branching for higher  $\lambda$  influences more the number of *actual* transitions taken (i.e. transitions that are not the added self-loops). As a result, the advantage of the prophetic scheduler is only little as (i) it boils down to observing the outcome of a less and less random choice and (ii) the observed quantity has little impact on how many actual transitions are taken.

As a result of Lemma 4, we obtain that Algorithm 1 terminates. Its correctness follows from Lemma 1, 2 and 3, all summarized by the following theorem.

**Theorem 1.** *Algorithm 1 computes an approximation of  $\text{val}^\nabla$  up to error  $\varepsilon$ .*

*Remark 2.* Algorithm 1 determines a sufficiently large  $\lambda$  in an exponential search fashion. In practice, this approach is efficient w.r.t. the total number  $I$  of *iterations* needed, i.e. the total number of times  $\underline{v}_k$  and  $\bar{w}_k$  are computed from  $\underline{v}_{k+1}$  and  $\bar{w}_{k+1}$ . Namely, in practice the error monotonously decreases when the rate increases (not in theory but we never encountered the opposite case on our extensive experiments.) As a result,  $\lambda$  found by Algorithm 1 satisfies  $\lambda < 2 \cdot \lambda^*$  where  $\lambda^*$  is the minimal sufficiently large rate. As the number of iterations needed for one approximation is linear in the uniformisation rate used, we have  $I = 2I_\lambda < 4 \cdot I_{\lambda^*}$ , where each  $I_{\lambda'}$  denotes the number of iterations needed for the computation for the fixed rate  $\lambda'$ .

### 3.4 Extracting the scheduler

By computing the lower bound, Algorithm 1 also produces [3] an untimed scheduler  $\sigma_\lambda^\nabla$  that is  $\varepsilon$ -optimal on the *uniformised* model  $\mathcal{C}_\lambda^\nabla$ . In the *original* CTMDP  $\mathcal{C}$ , we cannot use  $\sigma_\lambda^\nabla$  directly as its choices are tailored to the high rate  $\lambda$ . We can however use a *stochastic update* scheduler attaining the same value. Informally, a (timed) *stochastic update scheduler*  $\sigma = (\mathcal{M}, \sigma_u, \pi_0)$  operates over a countable set  $\mathcal{M}$  of memory elements where the initial memory value is chosen randomly according to the distribution  $\pi_0$  over  $\mathcal{M}$ . The stochastic update function  $\sigma_u$ , given the current memory element, state, and the time spent there, defines a distribution specifying the action to take and how to update the memory. Intuitively, the stochastic update is used for simulating the high-rate transitions that would be taken in  $\mathcal{C}_\lambda^\nabla$ ; their total count so far is stored in the memory. For a formal definition of stochastic update and the construction, see the Appendix.

**Lemma 5.** *The values  $(\underline{v}_k)_{0 \leq k \leq N}$  computed by Algorithm 1 for given  $\mathcal{C}$ ,  $\nabla$ , and  $\varepsilon > 0$  yield a stochastic update scheduler  $\tilde{\sigma}_{TD}^\nabla$  that is  $\varepsilon$ -optimal in  $\mathcal{C}$ .*



## 4 Existing Algorithms

This section briefly reviews the various published algorithms solving Problem 1. In contrast to Algorithm 1 (called UNIF<sup>+</sup> or U<sup>+</sup> for short), they all discretise time into a finite number of time points  $t_0, t_1, \dots, t_n$  where  $t_0 = 0$  and  $t_n = T$ . They iteratively approximate the values  $\text{val}^\nabla(s; t_i) := \sup_{\sigma \in \text{Tim}_\nabla} \Pr_\sigma^s [\Diamond^{\leq t_i} G]$  when  $t_i$  time units remain at state  $s$ . Three different iteration concepts have been proposed, each approximating  $\text{val}^\nabla(s; t_{i+1})$  from approximations of  $\text{val}^\nabla(s'; t_i)$ .

*Exponential approximation – early* [26,15]. Assuming equidistant points  $t_i$  one can approximate the (early) value function by piece-wise exponential functions. A  $k$ -order approximation considers only runs where at most  $k$  steps are taken between any two time points. This can yield an a priori error bound. The higher  $k$ , the less time points are required for a given precision, but the more computation is needed per time point. We refer to these algorithms by EXPSTEP- $k$  or ES- $k$  for short. Only ES-1 [26] and ES-2 [15] have been implemented so far.

*Polynomial approximation – late* [9]. Another way to approximate the (late) value function on equidistant time points uses polynomials. As before, the higher the degree of the polynomials, the higher is the computational effort, but the number of discretised time points required to assure an a priori error bound decreases. We call these algorithms POLYSTEP- $k$  or PS- $k$  in the sequel, only PS-1, PS-2, and PS-3 have been implemented. Among these, PS-2 has better worst-case behaviour, but PS-3 has been reported to often perform better in practice.

*Adaptive discretisation – late* [7]. This approach is not based on an a priori error bound but instead computes both under- and over-approximations of the values  $\text{val}^\nabla(s; t_i)$ . This allows one to lay out the time points *adaptively*. Depending on the shape of the value function, the time step can be prolonged until the error allowed for this step is reached. This greatly reduces the number of time points, relative to the worst case. We refer to this algorithm as ADAPTSTEP or AS.

## 5 Empirical Evaluation and Comparison

In this section we present an exhaustive empirical comparison of the different algorithmic approaches discussed.

**Benchmarks.** The experiments are performed on a diverse collection of published benchmark models. This collection is the first of its kind for CTMDP, as far as we know and contains the following parametrised models:

**PS- $K$ - $J$**  The *Polling System* case [12,30] consists of two stations and one server. Incoming requests of  $J$  types are buffered in two queues of size  $K$  each, until they are processed by the server and delivered to their station. We consider the undesirable states with both queues being full to form the goal state set.

- QS- $K$ - $J$**  The *Queuing System* [14] stores requests of  $J$  different types into two queues of size  $K$ . Each queue is attached to a server. Two servers fetch requests from their corresponding queues and process them. One of them can non-deterministically decide to insert a request after processing into the other server's queue. Goal states are again those with both queues full.
- DPMS- $K$ - $J$**  The *Dynamic Power Management System* [27] is a CTMDP model of the internals of a Fujitsu disk drive. The model consists of four components: service requester (SR), service queue (SQ), service provider (SP), and power manager (PM). SR generates tasks of  $J$  types differing in energy demand that are buffered by the queue SQ of size  $K$ . Afterwards they are delivered to SP to be processed. SP can work in different modes ranging from sleep and stand-by to full processing mode, selected by PM. We define a state as goal if the queue of at least one task type is full.
- GFS- $N$**  The *Google File System* [10,11] splits files into chunks of equal size, each chunk is maintained by one of  $N$  chunk servers. We fix the number of chunks a server may store to 5 000 and the total number of chunks to 100 000. While other benchmarks start in optimal conditions, the GFS starts in the broken state where no chunk is stored. A state is defined as goal if the system is back up and for each chunk at least one copy is available.
- FTWC- $N$**  The *Fault Tolerant Workstation Cluster* [16], originally described by a GSPN, models two networks of  $N$  workstations each, interconnected by a switch. The two switches communicate via a backbone. Workstations, switches, and the backbone fail after exponentially distributed delays, and can be repaired only one at a time. We define a state as goal if in total less than  $N$  workstations are operational and connected to each other.
- SJS- $M$ - $J$**  The *stochastic job scheduling* [5] models a multiprocessor architecture running a sequence of independent jobs. It consists of  $M$  identical processors and  $J$  jobs, where each job's service time is governed by an exponential distribution. As goal we define the desirable states with all jobs completed.
- ES- $K$ - $R$**  The *Erlang Stages* is a synthetic model with known characteristics [31]. It has two different paths to reach the goal state: a fast but risky path or a slow but sure path. The slow path is an Erlang chain of length  $K$  and rate  $R$ .

**Implementation aspects.** Unbiased performance evaluation of algorithms originally developed by different researchers is not easy even with all original implementations at hand. Namely, they may use different programming languages or rely on different platforms with incomparable performance and memory management. However, reimplementing a published algorithm may induce unfairness as the original implementation may use specific data structures or other optimisations that go beyond what is explained in the respective publication.

We adapted/implemented all algorithms in C/C++, trying to avoid the shortcomings. We used a common infrastructure from the IMCA/MAMA toolset [12]. Thus, we could directly use the original IMCA implementations of EXPSTEP-1 and of EXPSTEP-2 [15]. The original implementation [6] of ADAPT-STEP in MRMC [19] needed only minor adaptations, as MRMC uses a data structure identical to ours. Finally, for POLYSTEP, we closely followed the original Java code [9]. Our C version clearly outperforms the original Java version.

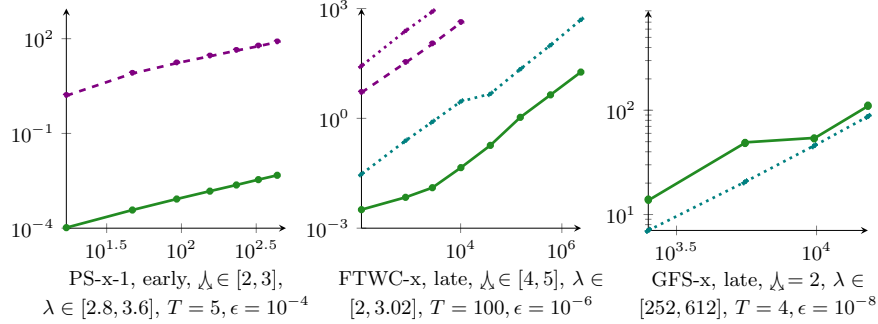


Fig. 1: Selected experiments: Increasing state space size.

We implemented all algorithms with standard double precision arithmetic, observing no issues with numerical stability in our experiments. All values computed by different algorithms lie within the expected precision from each other.

We used parameter values  $k^{\max} = 10$  and  $\omega = 0.1$  for AS, as recommended. We always ran both adaptive and non-adaptive variant of AS and display the better results (mostly adaptive). Based on our tests, we fixed  $\kappa := 0.1$  for  $\mathcal{U}^+$ .

**Empirical Results.** In this section we present our empirical observations. We consider early and late scheduling problems separately (because the encoding mentioned in Remark 1 of Section 2, is exponential); only  $\mathcal{UNIF}^+$  can be directly run on both problems. All experiments were run on a single core of Intel Core i7-4790 with 16GB of RAM, computing a total of about 2350 data points.

The memory requirements of all the considered algorithms do not deviate considerably and thus are not reported. This echoes that all space complexities are linear in the model size. We encountered no significant impact of additional dependencies of POLYSTEP on a hidden model parameter (number of “switching points”, coarsely bounded in [9]).

In the following, we focus on the time requirements. We first show plots of a few selected experiments that represent well our general observations. Later, we give a short summary of all experiments. All plots presented below use logarithmic scale for the runtime (in seconds). Some data points are missing as we applied a time limit of 15 minutes for every computation and also because the original implementation of EXPSTEP-2 cannot handle models with more than two actions per state. We use symbol  $\lambda_{\mathcal{A}}$  to denote the maximal number of action choices and  $\lambda$  for the maximal exit rate. We use the symbol “x” whenever the varying parameter is a part of the model name, e.g. PS-2-x.

**State space.** In Figure 1 we illustrate the effect of enlarging the state space.

On the left there is a plot for early algorithms representing the general trend:  $\mathcal{UNIF}^+$  outperforms EXPSTEP-1 (as well as EXPSTEP-2 where applicable). For late algorithms in the plots on the right, the situation is more diverse, with  $\mathcal{UNIF}^+$  and ADAPTSTEP outperforming the POLYSTEP algorithms. All algorithms exhibit similar dependency on the growth of the state space.

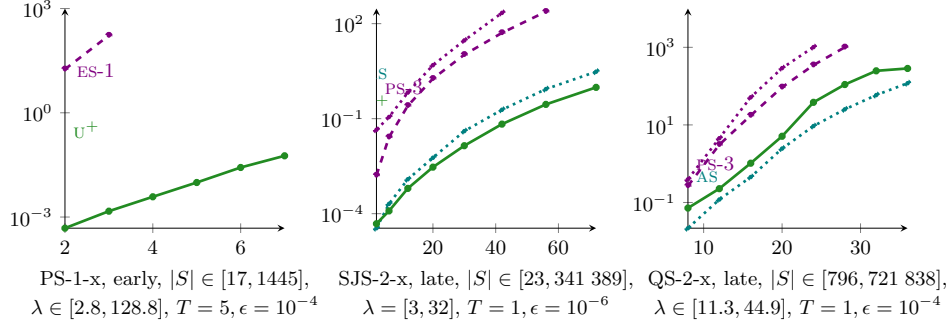


Fig. 2: Selected experiments: Increasing number of action choices.

**Action choices.** Figure 2 displays the effect of increasing the number of actions to choose from. For early schedulers (left) UNIF<sup>+</sup> generally dominates EXPSTEP-1. For late schedulers, again UNIF<sup>+</sup> and ADAPTSTEP dominate POLYSTEP. Increasing the choice options in our models generally induces larger state spaces, so the observed growth is not to be attributed to the computational difficulty resulting from an increase in choice options alone.

**Precision.** Figure 3 details precision dependency. Across all models, UNIF<sup>+</sup> works very well, excepts for some high precision cases, such as the DPMS

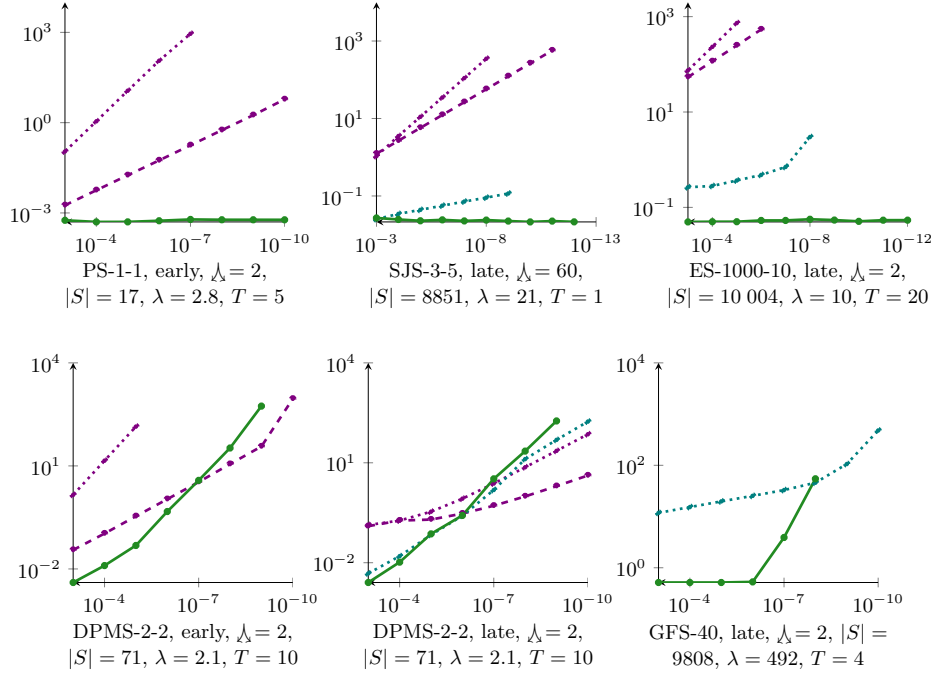


Fig. 3: Selected experiments: Increasing precision.

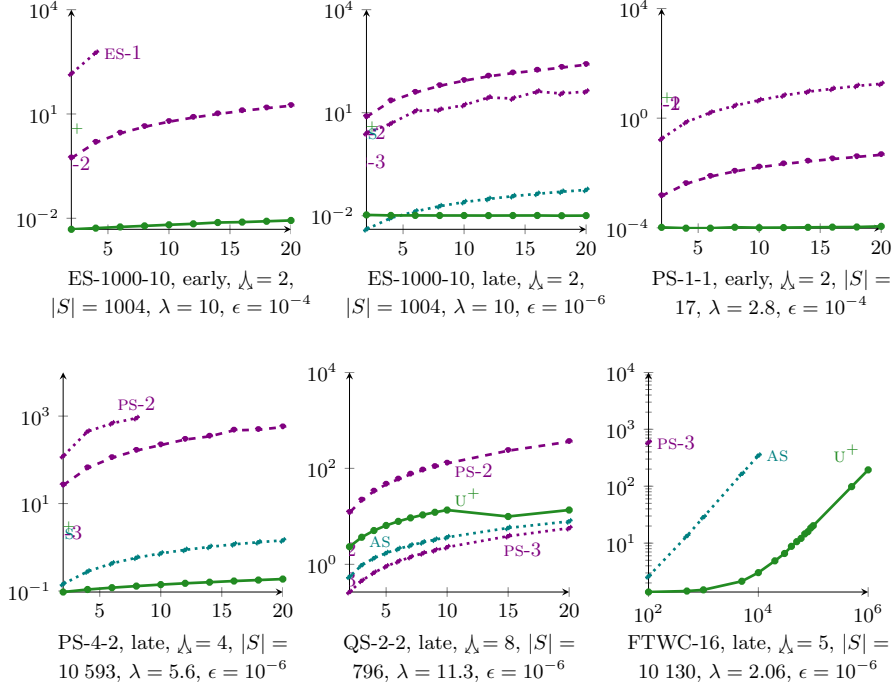
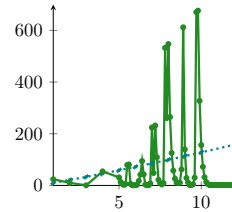


Fig. 4: Selected experiments: Increasing time bound.

models, where EXPSTEP-2 might be preferable over UNIF<sup>+</sup> in the early setting (bottom left), and similarly for ADAPTSTEP in the late setting (bottom middle). The same is true for the GFS case (bottom right). On the other hand, for some models (examples in the first row) UNIF<sup>+</sup> delivers very high precision without any runtime increase. It is also interesting that generally the sensitivity of all algorithm to required precision is more than linear in the number of precision digits.

**Time bound.** Figure 4 illustrates the effect of increasing the time bound. Again, the UNIF<sup>+</sup>-algorithm is the least sensitive in the early setting. For late scheduling, there are some notable QS instances where POLYSTEP-3 outperforms both ADAPTSTEP and UNIF<sup>+</sup> (bottom middle). Very large time bounds make sense only for a few models (bottom right, log-log-scale). Elsewhere, the values converge making it trivial for AS and U<sup>+</sup>.

Among the many instances we considered we found a few instances where the late UNIF<sup>+</sup>-algorithm shows surprising sensitivity to changes in time bound, particularly for high precision scenarios. This is exemplified on the right (GFS, late,  $\Delta = 2$ ,  $|S| = 9808$ ,  $\lambda = 492$ ,  $\epsilon = 10^{-8}$ , increasing time bound, no log scale). In line with the apparent general tendency of the algorithms for increasing parameter values, the work and thus time needed tends to increase monotonously.



	max. $ S $	max. $\Delta$	max. exit rate range	best in early (# of cases)	best in late (# of cases)
PS:	743 969	7	5.6 – 129.6	$U^+$ ( <b>32</b> )	$U^+$ ( <b>47</b> )
QS:	16 924	36	6.5 – 44.9	$U^+$ ( <b>32</b> )	PS-3( <b>18</b> ), $U^+$ (17), AS (15)
DPMS:	366 148	7	2.1 – 9.1	$U^+$ ( <b>31</b> ), ES-2(3), N/A(1)	AS ( <b>24</b> ), $U^+$ (14), PS-3(6)
GFS:	15 258	2	252 – 612	$U^+$ ( <b>40</b> )	AS ( <b>23</b> ), $U^+$ (11)
FTWC:2	373 650	5	2 – 3.02	$U^+$ ( <b>25</b> )	$U^+$ ( <b>32</b> )
SJS:	18 451	72	3 – 32	$U^+$ ( <b>57</b> ), ES-2(2)	$U^+$ ( <b>70</b> ), AS (29)
ES:	30 004	2	10	$U^+$ ( <b>23</b> ), ES-2(4), N/A(1)	$U^+$ ( <b>28</b> ), PS-3(2)

Table 1: Overview of experiments summarising which algorithm performed best how many times; N/A indicates that no algorithm completed within 15 minutes.

Instead, small variations in time bound may lead to great savings in runtime for  $UNIF^+$ . This is rooted in the error calculated while running the algorithm coincidentally falling into the allowed margin. Less extreme examples of this behaviour are included in Figure 3 top row and Figure 4 bottom middle. We observed such time savings only for  $UNIF^+$ , not for any other algorithm, though conceptually the runtime of ADAPTSTEP might profit from similar effects as well. The exact conditions of this behaviour are still to be found.

A complete list of model files, additional statistics, result tables as well as all prototype implementations are available at the following URL:

<http://depend.cs.uni-saarland.de/~hahate/atva15/>

**Evaluation and Discussion.** The results presented show that a general answer about the relative performance of the proposed algorithms is not easy to give, but appears very much dependent on model parameters outside the awareness of the modeller. Thus there is no clear winner across all models. Still, our benchmarking, summarised in Table 1, provides some general insights:

- All algorithms are naturally sensitive to increases in model parameters. Their runtime mostly behaves linear in the time bounds and the state space size, exponential in precision and superlinear (though still polynomial) in fanout.
- For early schedulers EXPSTEP-1 is not competitive.  $UNIF^+$  mostly outperforms EXPSTEP-2.
- For late schedulers POLYSTEP-1 is not competitive and POLYSTEP-3 is effectively faster than POLYSTEP-2. ADAPTSTEP and  $UNIF^+$  mostly outperform POLYSTEP-3. Still each of the late algorithms  $\{ADAPTSTEP, UNIF^+, POLYSTEP-3\}$  is dominating the other two on at least one model instance. The particular algorithmic strengths have no obvious relation to model parameters available to the modeller.
- For low precision,  $UNIF^+$  appears to be the preferred choice. For high precision, ADAPTSTEP is a more stable choice than  $UNIF^+$ . Yet its performance depends on non-obvious model particularities and algorithm parameters.

All in all,  $UNIF^+$  is easy to implement for both early and late, and competitive across a wide range of models. In settings where an a posteriori error bound is

enough, a good approximation can be usually obtained by a variant of  $\text{UNIF}^+$  that computes only the first iteration and does not increase the uniformisation rate (see the accompanying web for the error bounds obtained in experiments).

## 6 Conclusion

This paper has introduced  $\text{UNIF}^+$ , a new and simple algorithm for time-bounded reachability objectives in CTMDPs. We studied this and all other published algorithms in an extensive comparative evaluation for both early and late scheduling. In general,  $\text{UNIF}^+$  performs very well across the benchmarks, apart from late scheduling and high precision, where it appears hard to predict which of the algorithms  $\text{UNIF}^+$ ,  $\text{ADAPTSTEP}$ ,  $\text{POLYSTEP-3}$  performs best. One might consider to follow an approach inspired by the distributed concurrent solver in  $\text{GUROBI}$  [13]. The idea is to launch all three implementations to run concurrently on distinct cores and report the result as soon as the first one terminates.

For researchers who want to extend an existing CTMC model checker to a CTMDP model checker, the obvious choice is the  $\text{UNIF}^+$ -algorithm: It works right away for early and for late optimisation, and it requires only a small change to the uniformisation subroutine used at the core of CTMC model checking.

**Acknowledgements** We are grateful to Moritz Hahn (ISCAS Beijing), Dennis Guck (Universiteit Twente), and Markus Rabe (UC Berkeley) for discussions and technical contributions. This work is supported by the EU 7th Framework Programme projects 295261 (MEALS) and 318490 (SENSATION), by the Czech Science Foundation project P202/12/G061, the DFG Transregional Collaborative Research Centre SFB/TR 14 AVACS, and by the CDZ project 1023 (CAP).

## References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Verifying continuous time Markov chains. In: CAV. pp. 269–276 (1996)
2. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.: Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Software Eng.* 29(6), 524–541 (2003)
3. Baier, C., Hermanns, H., Katoen, J., Haverkort, B.R.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theor. Comput. Sci.* 345(1), 2–26 (2005)
4. Brázdil, T., Forejt, V., Krcál, J., Kretínský, J., Kucera, A.: Continuous-time stochastic games with time-bounded reachability. *Inf. Comput.* 224, 46–70 (2013)
5. Bruno, J.L., Downey, P.J., Frederickson, G.N.: Sequencing tasks with exponential service times to minimize the expected flow time or makespan. *J. ACM* 28(1), 100–113 (1981)
6. Buchholz, P., Hahn, E.M., Hermanns, H., Zhang, L.: Model checking algorithms for CTMDPs. In: CAV. pp. 225–242 (2011)
7. Buchholz, P., Schulz, I.: Numerical analysis of continuous time Markov decision processes over finite horizons. *Computers & OR* 38(3), 651–659 (2011)

8. Eisentraut, C., Hermanns, H., Katoen, J., Zhang, L.: A semantics for every GSPN. In: *Petri Nets 2013*. pp. 90–109 (2013)
9. Fearnley, J., Rabe, M., Schewe, S., Zhang, L.: Efficient Approximation of Optimal Control for Continuous-Time Markov Games. In: *FSTTCS*. pp. 399–410 (2011)
10. Ghemawat, S., Gobioff, H., Leung, S.T.: The Google file system. In: *SOSP*. pp. 29–43. ACM (2003)
11. Guck, D.: Quantitative Analysis of Markov Automata. Master’s thesis, RWTH Aachen University (June 2012)
12. Guck, D., Hatefi, H., Hermanns, H., Katoen, J.P., Timmer, M.: Modelling, reduction and analysis of Markov automata. In: *QEST*. pp. 55–71 (2013)
13. Gurobi Optimization, Inc.: Gurobi optimizer reference manual, version 6.0 (2015)
14. Hatefi, H., Hermanns, H.: Model checking algorithms for Markov automata. *ECE-ASST* 53 (2012)
15. Hatefi, H., Hermanns, H.: Improving time bounded reachability computations in interactive Markov chains. In: *FSEN*. pp. 250–266 (2013)
16. Haverkort, B.R., Hermanns, H., Katoen, J.: On the use of model checking techniques for dependability evaluation. In: *SRDS 2000*. pp. 228–237. IEEE CS (2000)
17. Hermanns, H., Katoen, J.: The How and Why of interactive Markov chains. In: *FMCO 2009*. pp. 311–337 (2009)
18. Jensen, A.: Markoff chains as an aid in the study of Markoff processes. *Scandinavian Actuarial Journal* 1953, 87–91 (1953)
19. Katoen, J., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.* 68(2), 90–104 (2011)
20. Lefèvre, C.: Optimal control of a birth and death epidemic process. *Operations Research* 29(5), 971–982 (1981)
21. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons (1994)
22. Meyer, J.F., Movaghar, A., Sanders, W.H.: Stochastic activity networks: Structure, behavior, and application. In: *PNPM*. pp. 106–115 (1985)
23. Miller, B.L.: Finite state continuous time Markov decision processes with a finite planning horizon. *SIAM Journal on Control* 6(2), 266–280 (1968)
24. Neuhäuffer, M.R.: Model checking nondeterministic and randomly timed systems. Ph.D. thesis, RWTH Aachen University (2010)
25. Neuhäuffer, M.R., Stoelinga, M., Katoen, J.: Delayed nondeterminism in continuous-time Markov decision processes. In: *FOSSACS*. pp. 364–379 (2009)
26. Neuhäuffer, M.R., Zhang, L.: Time-bounded reachability probabilities in continuous-time Markov decision processes. In: *QEST*. pp. 209–218 (2010)
27. Qiu, Q., Qu, Q., Pedram, M.: Stochastic modeling of a power-managed system-construction and optimization. *IEEE Trans. on CAD of Integrated Circuits and Systems* 20(10), 1200–1217 (2001)
28. Rabe, M.N., Schewe, S.: Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games. *Acta Inf.* 48(5-6), 291–315 (2011)
29. Rabe, M.N., Schewe, S.: Optimal time-abstract schedulers for CTMDPs and continuous-time Markov games. *Theor. Comput. Sci.* 467, 53–67 (2013)
30. Timmer, M., van de Pol, J., Stoelinga, M.: Confluence reduction for Markov automata. In: *FORMATS*. pp. 243–257 (2013)
31. Zhang, L., Neuhäuffer, M.R.: Model checking interactive Markov chains. In: *TACAS*. pp. 53–68 (2010)



## A Proofs from Section 3

We first prove the following auxiliary lemma characterizing the functions that are used to approximate the lower and upper bounds.

**Lemma A.1.** *For every  $0 \leq k < N$  and  $s \in S$ , we have*

$$\begin{aligned}\underline{v}_k(s) &= \sup_{\sigma \in \text{Unt}} \sum_{i=k}^{N-1} \Pr_{\sigma}^{s_{in}} \left[ \Diamond_{\leq i}^{\leq T} G \mid s @ k \right] \\ \bar{v}_k(s) &= \sum_{i=k}^{N-1} \sup_{\sigma \in \text{Unt}} \Pr_{\sigma}^{s_{in}} \left[ \Diamond_{\leq i}^{\leq T} G \mid s @ k \right] = \sum_{i=k}^{N-1} \sup_{\sigma \in \text{Tim}_{\nabla}} \Pr_{\sigma}^{s_{in}} \left[ \Diamond_{\leq i}^{\leq T} G \mid s @ k \right] \\ \bar{w}_k(s) &= \sup_{\sigma \in \text{Unt}} \Pr_{\sigma}^{s_{in}} [\Diamond_{\leq N-1} G \mid s @ k] = \sup_{\sigma \in \text{Tim}_{\nabla}} \Pr_{\sigma}^{s_{in}} [\Diamond_{\leq N-1} G \mid s @ k]\end{aligned}$$

where  $s_{in}$  is the initial state,  $s @ k$  are the runs that visit  $s$  after  $k$  steps and do not reach  $G$  before  $k$  steps, and  $\Diamond_{\leq N-1} G$  are the runs that reach  $G$  within  $N-1$  steps taken in arbitrary time.

*Proof.* Let us first assume  $s \in G$ . We have  $\bar{v}_k(s) = \underline{v}_k(s) = \sum_{i=k}^{N-1} \psi_{\lambda}(i)$  and  $\bar{w}_k(s) = 1$  which is also equal to the right hand sides of the equalities above. Next, we prove the equalities for  $s \notin G$  by induction. First, the first and only summand in the right hand sides above equals to 0 while also  $\bar{v}_{N-1}(s) = \underline{v}_{N-1}(s) = 0$ . Next, let  $k < N-1$  assuming the equalities above for  $k+1$ .

$$\begin{aligned}\underline{v}_k(s) &= \max_a \sum_{s'} \mathbf{P}_{\lambda}^{\nabla}(s, a, s') \cdot \underline{v}_{k+1}(s') \\ &= \max_a \sum_{s'} \mathbf{P}_{\lambda}^{\nabla}(s, a, s') \sup_{\sigma \in \text{Unt}} \sum_{i=k+1}^{N-1} \Pr_{\sigma}^{s_{in}} \left[ \Diamond_{\leq i}^{\leq T} G \mid s' @ k + 1 \right] \\ &= \sup_{a, \sigma \in \text{Unt}} \sum_{i=k+1}^{N-1} \sum_{s'} \mathbf{P}_{\lambda}^{\nabla}(s, a, s') \Pr_{\sigma}^{s_{in}} \left[ \Diamond_{\leq i}^{\leq T} G \mid s' @ k + 1 \right] \\ &= \sup_{\sigma \in \text{Unt}} \sum_{i=k+1}^{N-1} \Pr_{\sigma}^{s_{in}} \left[ \Diamond_{\leq i}^{\leq T} G \mid s @ k \right] \\ &= \sup_{\sigma \in \text{Unt}} \sum_{i=k}^{N-1} \Pr_{\sigma}^{s_{in}} \left[ \Diamond_{\leq i}^{\leq T} G \mid s @ k \right]\end{aligned}$$

since the first summand equals to zero. Similarly for  $\bar{w}_k(s)$ , we have

$$\begin{aligned}\bar{w}_k(s) &= \max_a \sum_{s'} \mathbf{P}_{\lambda}^{\nabla}(s, a, s') \bar{w}_{k+1}(s') \\ &= \max_a \sum_{s'} \mathbf{P}_{\lambda}^{\nabla}(s, a, s') \sup_{\sigma \in \text{Unt}} \Pr_{\sigma}^{s_{in}} [\Diamond_{\leq N-1} G \mid s' @ k + 1] \\ &= \sup_{\sigma \in \text{Unt}} \Pr_{\sigma}^{s_{in}} [\Diamond_{\leq N-1} G \mid s @ k]\end{aligned}$$

and the same hold when ranging over schedulers in  $Tim_{\nabla}$ . Finally, for  $\bar{v}_k(s)$ ,

$$\begin{aligned}
\bar{v}_k(s) &= \sum_{i=k}^{N-1} \psi_{\lambda}(i) \cdot \bar{w}_{(N-1)-(i-k)}(s) \\
&= \sum_{i=k}^{N-1} \psi_{\lambda}(i) \cdot \sup_{\sigma \in Unt} \Pr_{\sigma}^{s_{in}} [\Diamond_{\leq N-1} G \mid s @ (N-1) - i + k] \\
&= \sum_{i=k}^{N-1} \psi_{\lambda}(i) \cdot \sup_{\sigma \in Unt} \Pr_{\sigma}^{s_{in}} [\Diamond_{\leq i} G \mid s @ k] \\
&= \sum_{i=k}^{N-1} \sup_{\sigma \in Unt} \Pr_{\sigma}^{s_{in}} [\Diamond_{\leq i}^T G \mid s @ k]
\end{aligned}$$

and again the same hold when ranging over schedulers in  $Tim_{\nabla}$ .  $\square$

**Lemma 2.** It holds that  $\text{val}^e \leq \text{val}^{\ell}$ , and for any CTMDP  $\mathcal{C}_{\lambda}^{\nabla}$ ,  $\underline{\text{val}} \leq \text{val}^{\nabla} \leq \bar{\text{val}}$ .

*Proof.*  $\text{val}^e \leq \text{val}^{\ell}$  follows directly from the fact that  $Tim_e \subseteq Tim_{\ell}$ .  $\underline{\text{val}} \leq \bar{\text{val}}$  since

$$\sup_{\sigma \in Tim_{\nabla}} \sum_{i=0}^{\infty} \Pr_{\sigma}^s [\Diamond_{\leq i}^T G] \leq \sum_{i=0}^{\infty} \sup_{\sigma \in Tim_{\nabla}} \Pr_{\sigma}^s [\Diamond_{\leq i}^T G]$$

(optimizing for each subset separately yields higher value).

Furthermore  $Unt \subseteq Tim_e$  implies  $\underline{\text{val}} \leq \text{val}^e$ , and for each  $i \in \mathbb{N}_0$ , it follows from Lemma A.1 that

$$\sup_{\sigma \in Tim_{\nabla}} \Pr_{\sigma}^s [\Diamond_{\leq i}^T G] = \sup_{\sigma \in Unt} \Pr_{\sigma}^s [\Diamond_{\leq i}^T G].$$

**Lemma 3.** In any CTMDP  $\mathcal{C}_{\lambda}^{\nabla}$ ,  $\|\underline{v}_0 - \underline{\text{val}}\|_{\infty} \leq \varepsilon \cdot \kappa$  and  $\|\bar{v}_0 - \bar{\text{val}}\|_{\infty} \leq \varepsilon \cdot \kappa$ .

*Proof.* The proof follows from Lemma A.1. It is completed by the observation [4] that the probability of  $\geq N$  steps to be taken within  $T$  is  $\leq \varepsilon \cdot \kappa$ . This is independent of the scheduler and hence, for both  $\underline{v}_0$  and  $\bar{v}_0$  we obtain the desired error bound.  $\square$

**Lemma 4.** We have  $\lim_{\lambda \rightarrow \infty} g_{\lambda} \rightarrow 0$  where  $g_{\lambda}$  denotes the gap  $\|\underline{\text{val}} - \bar{\text{val}}\|_{\infty}$  in  $\mathcal{C}_{\lambda}^{\nabla}$ .

*Proof.* From Lemmata 2 and 1 we have that for any  $\lambda$ ,  $\underline{\text{val}} \leq \text{val}^{\nabla} \leq \bar{\text{val}}$ . It remains to show that for any state  $s \in S$ , we have

$$\lim_{\lambda \rightarrow \infty} |\underline{\text{val}}(s) - \bar{\text{val}}(s)| \rightarrow 0. \tag{1}$$

Let  $\lambda_0$  be the maximal exit rate in  $\mathcal{C}$  and let  $\varepsilon > 0$ . We need to find a uniformisation rate  $\lambda = k\lambda_0$  such that  $|\underline{\text{val}}(s) - \overline{\text{val}}(s)| \leq \varepsilon$ . Consider Chebyshev inequality

$$Pr[|\psi_{\lambda T} - \lambda T| \geq m\sigma] \leq \frac{1}{m^2}$$

where  $\sigma$  is standard deviation of  $\psi_{\lambda T}$  and  $m > 0$ . Let  $\frac{1}{m^2} = \frac{\varepsilon}{6}$ . Then Chebyshev inequality for  $\psi_{\lambda T}$  can be written as

$$Pr[|\psi_{\lambda T} - \lambda T| \geq \sqrt{\frac{6\lambda T}{\varepsilon}}] \leq \frac{\varepsilon}{6}$$

or

$$Pr[\psi_{\lambda T} \in (\lambda T - \sqrt{\frac{6\lambda T}{\varepsilon}}, \lambda T + \sqrt{\frac{6\lambda T}{\varepsilon}})] > 1 - \frac{\varepsilon}{6}$$

For any uniformisation rate  $\lambda = k \cdot \lambda_0$ , we define  $a_\lambda = \lfloor \lambda T - \sqrt{6\lambda T/\varepsilon} \rfloor$  and  $b_\lambda = \lceil \lambda T + \sqrt{6\lambda T/\varepsilon} + 1 \rceil$ . Then,

$$\sum_{i=a_\lambda}^{b_\lambda-1} \psi_\lambda(i) > 1 - \frac{\varepsilon}{6}. \quad (2)$$

In the uniformised model  $\mathcal{C}_{k \cdot \lambda_0}^\nabla$ , the probability of not changing state in one step is

$$\frac{\lambda - E(s, \alpha)}{\lambda} \geq \frac{\lambda - \lambda_0}{\lambda} = \frac{k-1}{k}$$

Therefore, for  $c_\lambda = b_\lambda - a_\lambda \leq 2(\sqrt{6\lambda T/\varepsilon} + 1)$ , the probability  $p_\lambda$  of not changing state at all within  $c_\lambda$  steps satisfies

$$\begin{aligned} p_\lambda &= \left(\frac{k-1}{k}\right)^{c_\lambda} \geq \left(\frac{k-1}{k}\right)^{2(\sqrt{6k\lambda_0 T/\varepsilon} + 1)} = \\ &= \left[\left(\frac{k-1}{k}\right)^{\sqrt{k}}\right]^{2(\sqrt{6\lambda_0 T/\varepsilon})} \left(\frac{k-1}{k}\right)^2 = \\ &= \left[\left(1 - \frac{1}{\sqrt{k}}\right)^{\sqrt{k}} \left(1 + \frac{1}{\sqrt{k}}\right)^{\sqrt{k}}\right]^{2(\sqrt{6\lambda_0 T/\varepsilon})} \left(\frac{k-1}{k}\right)^2 \end{aligned}$$

Thus

$$\lim_{k \rightarrow \infty} p_\lambda = \left[\frac{1}{e} \cdot e\right]^{2(\sqrt{6\lambda_0 T/\varepsilon})} = 1 \quad (3)$$

Instead of (1) we prove that there is  $\lambda$  such that

$$|\overline{\text{v}}_0^{b_\lambda}(s) - \underline{\text{v}}_0^{b_\lambda}(s)| \leq \varepsilon/2.$$

where  $\underline{v}_i^{b_\lambda}(s)$ ,  $\overline{w}_i^{b_\lambda}(s)$ , and  $\overline{v}_i^{b_\lambda}(s)$  is defined for all  $k$  as  $\underline{v}$ ,  $\overline{w}$ , and  $\overline{v}$ , only replacing  $N$  by  $b_\lambda$ . This suffices, as from proof of Lemma 3, the approximations with upper bound  $b_\lambda$  are only more precise than approximations with upper bound  $N$ . Using (3), we fix  $\lambda$  to be such that  $p_\lambda \geq 1 - \varepsilon/6$ . Then, we have

$$\underline{v}_0^{b_\lambda}(s) \geq \underline{v}_{a_\lambda}^{b_\lambda}(s)$$

and from (2), we can obtain by straightforward induction

$$\geq \underline{u}_{a_\lambda}^{b_\lambda}(s) - \frac{\varepsilon}{6}$$

where  $\underline{u}_k$  is defined as  $\underline{v}_k$  except for goal states having value 1 instead of the sum of poisson probabilities. The term  $\underline{u}_{a_\lambda}^{b_\lambda}(s)$  now equals by definition the term  $\overline{w}_{(b_\lambda-1)-a_\lambda}^{b_\lambda}(s)$ , and hence

$$= \overline{w}_{(b_\lambda-1)-a_\lambda}^{b_\lambda}(s) - \frac{\varepsilon}{6} \geq \sum_{i=a_\lambda}^{b_\lambda-1} \psi_\lambda(i) \cdot \overline{w}_{(b_\lambda-1)-a_\lambda}^{b_\lambda}(s) - \frac{\varepsilon}{6};$$

Furthermore, from the fact that  $p_\lambda \geq 1 - \varepsilon/6$ , we obtain that each  $\overline{w}_{(b_\lambda-1)-i}^{b_\lambda}(s) \leq \overline{w}_{(b_\lambda-1)-a_\lambda}^{b_\lambda}(s) - \varepsilon/6$  and

$$\geq \sum_{i=a_\lambda}^{b_\lambda-1} \psi_\lambda(i) \cdot \overline{w}_{(b_\lambda-1)-i}^{b_\lambda}(s) - \frac{2\varepsilon}{6} \geq \sum_{i=0}^{b_\lambda-1} \psi_\lambda(i) \cdot \overline{w}_{(b_\lambda-1)-i}^{b_\lambda}(s) - \frac{3\varepsilon}{6}$$

and finally, we get by definition

$$= \overline{v}_0^{b_\lambda}(s) - \varepsilon/2.$$

□

Let  $\mathcal{D}(A)$  denote the set of all distributions over a discrete set  $A$ . Then

**Definition 5.** A stochastic update scheduler  $\sigma$  on a CTMDP  $\mathcal{C} = (S, \text{Act}, \mathbf{R})$  is a tuple  $\sigma = (\mathcal{M}, \sigma_u, \pi_0)$ , where

- $\mathcal{M}$  is a countable set of memory elements
- $\sigma_u : \mathcal{M} \times S \times \mathbb{R}_{\geq 0} \mapsto \mathcal{D}(\mathcal{M}, \text{Act})$  is the update function
- $\pi_0 : S \mapsto \mathcal{D}(\mathcal{M})$  distribution over initial memory values

The system operates under a stochastic update scheduler as follows. At first initial memory values are sampled from the distribution  $\pi_0(s)$ . Afterwards, given current memory value, current state and time spent in the state so far (not the time from the beginning of the process), the stochastic update function  $\sigma_u$  continuously updates the memory value and the action to be taken. When the system decides to leave the state, upon entering the successor state the memory is also updated by one.

**Lemma 5.** The values  $(\underline{v}_k)_{0 \leq k \leq N}$  computed by Algorithm 1 for given  $\mathcal{C}$ ,  $\nabla$ , and  $\varepsilon > 0$  yield a stochastic update scheduler  $\tilde{\sigma}_{TD}^\nabla$  that is  $\varepsilon$ -optimal in  $\mathcal{C}$ .

*Proof.* Let  $\mathcal{C} = (S, Act, \mathbf{R})$  be the original CTMDP,  $\lambda$  - the uniformization rate computed by Algorithm 1,  $\mathcal{C}_\lambda^\nabla = (S_\lambda, Act, \mathbf{R}_\lambda)$  - CTMDP uniformised with rate  $\lambda$ . Computation of the lower bound  $\underline{v}_0$  involves as well computation of the  $\varepsilon$ -optimal scheduler that attains the bound. Let  $\tilde{\sigma}_{U_{nt}}^\nabla : S_\lambda \times \mathbb{N}_{\geq 0} \mapsto Act$  be this scheduler and  $i\mathcal{C}_{U_{nt}}^\nabla = (iS_{U_{nt}}, i\mathbf{R}_{U_{nt}}^\nabla)$  - the CTMC induced by  $\tilde{\sigma}_{U_{nt}}^\nabla$ , where  $iS_{U_{nt}} = S_\lambda \times \mathbb{N}_0$ . Then,

$$i\mathbf{R}_{U_{nt}}(s_1, s_2) = \begin{cases} \mathbf{R}(s', \tilde{\sigma}_{U_{nt}}^\nabla(s', m), s'') & \text{if } s_1 = (s', m) \text{ and } s_2 = (s'', m+1) \\ \lambda - E(s', \tilde{\sigma}_{U_{nt}}^\nabla(s', m)) & \text{if } s_1 = (s', m) \text{ and } s_2 = (s', m+1) \\ 0 & \text{otherwise} \end{cases}$$

Let  $\pi_{(s,m)}(s', m+k, t)$  be the transient probability in  $i\mathcal{C}_{U_{nt}}^\nabla$  for state  $(s, m+k)$ , given that the system starts from state  $(s, m)$ . Then

$$\pi_{(s,m)}(s', m+k, t) = \begin{cases} e^{-\lambda t} \frac{(\lambda - E_0) \cdots (\lambda - E_{k-2})(\lambda - E_{k-1})}{k!} t^k & \text{if } s' = s \\ e^{-\lambda t} \frac{(\lambda - E_0) \cdots (\lambda - E_{k-2}) \mathbf{R}(s, \alpha_{k-1}, s')}{k!} t^k & \text{otherwise} \end{cases}$$

where  $\alpha_i = \tilde{\sigma}_{U_{nt}}^\nabla(s, m+i)$  and  $E_i = E(s, \alpha_i)$ .

W.l.o.g. we assume that after  $N$  transitions have been performed the scheduler  $\tilde{\sigma}_{U_{nt}}^\nabla$  takes the same decision for every state, irrespectively of the memory value. We denote this decision as  $\alpha_N(s)$ . We now define the finite memory stochastic update scheduler  $\tilde{\sigma}_{Tim} = (\mathcal{M}, \sigma_u, \pi_0)$ :

- $\mathcal{M} = [0..N] \cup \perp$
- $\forall m, m' \in \mathcal{M}, m \neq \perp, s \in S, t \in \mathbb{R}_{\geq 0}$

$$\sigma_u^\nabla(\perp, s, t) := [(\perp, \alpha_N(s)) \mapsto 1, \text{otherwise} \mapsto 0]$$

$$\sigma_u^\nabla(m, s, t)(m', a) := \begin{cases} \pi_{(s,m)}(s, m', t) & \text{if } m' \in [m+1..N] \text{ and} \\ & a = \tilde{\sigma}_{U_{nt}}^\nabla(s, m^\nabla), \text{ where} \\ & m^e = m, m^\ell = m' \\ \sum_{k=1}^{\infty} \pi_{(s,m)}(s, N+k, t) & \text{if } m' = \perp \text{ and} \\ & \nabla = e \text{ and } a = \tilde{\sigma}_{U_{nt}}^e(s, m) \text{ or} \\ & \nabla = \ell \text{ and } a = \alpha_N(s) \\ 0 & \text{otherwise} \end{cases}$$

$$- \forall s \in S \pi_0(s) := [0 \mapsto 1, \text{otherwise} \mapsto 0]$$

Intuitively, when the system moves to a state  $s$  and memory  $m$  has been collected up until this moment, it updates the memory according to the sub-process  $\mathcal{C}_\lambda^\nabla(s, m)$  of  $\mathcal{C}_\lambda^\nabla$  while residing in  $s$  and the memory update is finished when  $\mathcal{C}$  decides to leave  $s$ . This sub-process  $\mathcal{C}_\lambda^\nabla(s, m)$  is a process that starts when  $\mathcal{C}_\lambda^\nabla$  moves to the state  $s$  and evolves when the uniformized system takes introduced high-rate transitions. Thus, the value of the memory is equivalent to the length of the history of the uniformized process, i.e. the  $\tilde{\sigma}_{Tim}$  simulates evolution of the uniformized process and takes exactly the decisions that  $\tilde{\sigma}_{Unt}$  would take. Thus, the transient distribution of the processes induced by  $\tilde{\sigma}_{Tim}$  and  $\tilde{\sigma}_{Unt}$  are exactly the same.

The amount of memory used by  $\tilde{\sigma}_{Tim}$  is in  $O(N|S|)$ .